A Research Program in Advanced Information Systems

Final Report

by

Wallace E. Vander Velde

February 12, 1987

A RESEARCH PROGRAM IN ADVANCED INFORMATION SYSTEMS

Final Report

NASA Research Grant No. NAGW-448

June 1983 to September 1985

Principal Investigator: Wallace E. Vander Velde
Department of Aeronautics and
Astronautics
Massachusetts Institute of Technology
Cambridge, MA 02139

Wallace E Vander Velde
Wallace E. Vander Velde

February 12, 1987

A RESEARCH PROGRAM IN ADVANCED INFORMATION SYSTEMS

Introduction

This is the final report of the work accomplished under NASA Research Grant No. NAGW-448. This was a block grant, active over the period June 1, 1983 to September 30, 1985, which supported a number of research tasks dealing with computational issues of importance to NASA.

This program, conducted jointly by the Massachusetts Institute of Technology and the Charles Stark Draper Laboratory, Inc., was part of a NASA initiative in computer science and technology which reflects the growing importance of this discipline in aircraft and spacecraft design and operations relative to the traditional aerospace disciplines which NASA has long emphasized. The major objectives stated for this initiative were:

- Improve computer performance
- Enhance computer fault tolerance
- Assure correctness of executed software
- Extend our ability to deal with complexity

A stated purpose of the university portion of the program was to develop capability as well as research results. Toward that end, involvement of students in research activities, interactions among departments and research laboratories, and cooperation among universities, industry, and the NASA centers were specifically encouraged.

Description of the Program

MIT and CSDL organized a program of research which addresses all of the objectives cited above. It involved personnel from three departments at MIT and two groups at CSDL. The research topics were as follows:

- Identification of Large Space Structure Dynamics
- Special-Purpose Architectures for CFD
- Fault-Tolerant Processor Architectures
- Dataflow Techniques
- Software Specification and Verification Tools
- Management of Software Development
- The Software Environment for Concurrent Computing
- Parallel Algorithms and Architectures for the Solution of Partial Differential Equations

The objective of improved computer performance is addressed by the research task on Fault-Tolerant Processor Architectures, which dealt with computer configurations which offer high throughput as well as fault tolerance, and by the task on

Concurrent Computing. The objective of computer fault tolerance is addressed by the research on Fault-Tolerant Processor Architectures and the task on Dataflow Techniques. One of the motivations for this work on Dataflow was the potential of that methodology for specifying fault-tolerant computational structures. The objective of software correctness is addressed by the research task on Software Specification and Verification Tools, and indirectly by the task on Software Development Management, as one of the principal management questions is the issue of personnel assignments during the software debugging process. The objective of improved ability to deal with complexity is addressed by the research task on Special-Purpose Architectures for CFD and the task on Parallel Algorithms for the Solution of Partial Differential Equations. These computations are among the most complex that arise in the design and operation of aircraft and spacecraft.

The motivation for the research task on Identification of Large Space Structure Dynamics was to use this important function as a reference for the work on computer architectures. System identification is perhaps the most computer intensive aspect of spacecraft control and thus it serves as an excellent requirement for the throughput capability of computer architectures which are also fault tolerant. Also, as software development projects become more and more complex, the question of how best to manage this process takes on increasing significance. It is not obvious, for example, at what stages of a program the infusion of additional personnel results in improved performance on a project which is running behind schedule. It is issues such as this which were investigated in the research task on Management of Software Development.

For the purposes of planning and reporting, two meetings were held which involved participants in the program at MIT and CSDL as well as the NASA Technical Monitor and interested people from some of the NASA Centers. The first was a kickoff meeting held at the MIT Endicott House in Dedham, Massachusetts, June 15 -17, 1983. The Centers represented at this meeting included MSFC, LaRC, JPL, KSC, and GSFC. The discussion at this meeting was limited primarily to the research activity planned under this program. The second meeting was held at Langley Research Center, August 29 -31, 1984. All of the Centers cited above were represented at this meeting as well, and in addition, Robert Voigt of ICASE was in attendance. About half of the agenda of this meeting was devoted to reports on computer science activities at the centers represented. This served the desire of the Program Technical Monitor for good communication among the interested people at the Centers, in Headquarters, and those participating in the research program.

Discussion of the Research Tasks

In this section, brief reports of the work accomplished in each of the research tasks are given. The responsible persons are identified and publications resulting from the work are cited. The publications are more fully identified in the Publications section which follows.

Identification of Large Space Structure Dynamics -- This research task was under the direction of Professor William S. Widnall of the MIT Department of Aeronautics and Astronautics. Assisting in the work was graduate student Research Assistant Janice Voss. The purpose of this task was to provide a realistic reference point for the computational requirements of spacecraft of the future.

Both NASA and the Air Force are looking forward to deploying very large assemblies in space in the years ahead. These may be large optical surfaces, large antennas, large arrays of solar cells, or large structures incorporating more than one of these elements. An example of the latter is the Space Station which NASA has identified as a priority objective. These assemblies will necessarily be light in weight and will have little inherent rigidity. The common concept is that these structures will require active control not only for station keeping and attitude control, but also for damping structural vibration and figure control.

Because of the low mass and limited strength of these structures it will not be possible to erect them on the ground in the 1 g field and perform the usual series of static and dynamic tests. One consequence of this is that upon deployment in orbit the dynamic properties of the structure will not be well known. An estimate of these properties will have been made by finite-element computations or other means, but it is to be expected that this a priori model will be in error by an amount sufficient to preclude the proper functioning of the active control system. Thus one must anticipate an initial period after the structure has been assembled during which on-line identification of the dynamic characteristics of the structure will be performed. During this period, a simple control system will be engaged which is robust with respect to modeling errors and is intended only to maintain loose control over the system and damp vibrations. It may not be possible to perform the primary missions of the spacecraft during this period. Hence there is a natural motivation to share the primary on-board data processor between the identification task in the initial time period and other functions -- notably scientific sensor data processing -- during most of the mission.

Of the several functions associated with control of a spacecraft, the one that imposes the greatest computational

burden is system identification. If high performance control of a flexible space structure is required, it will have to be based on an accurate, high-ordered model of the spacecraft dynamics. Such a model involves hundred of parameters, and the task of system identification is to estimate the values of those parameters based on a body of input-output data. This is a parameter estimation problem of a high order which has severe computational requirements.

Initial work concentrated on surveying the literature on system identification, choosing a form of model for the spacecraft dynamics, and selecting the basic form of estimator to use as a baseline. There is a large literature on system identification but little of it is specialized to the problem of large space structure identification. Various forms of state space models having the minimum number of parameters are available to model linearized system dynamics. Eventually, the modal form of state space model was selected because of its convenient transformation to an autoregressive, moving average input-output form used in the actual estimation process. Parameter estimators based on least squares and maximum likelihood theories were reviewed with particular emphasis on recursive implementations. The final choice of estimator was a lattice form of the recursive least squares filter. This formulation of the estimator has smaller computational requirements than other forms and lends itself well to approximations which trade reduced computational throughput for a longer estimation interval. It is also straightforward to organize the required computation in a form that allows concurrent processing in a bank of parallel computers. Exploring the utility of concurrent computing is one of the objectives of the NASA initiative in computer science and technology.

This research task was pursued only during the first year of the program. The work did not lead to publications during that time.

Special-Purpose Architectures for CFD -- Responsible for this research task was Professor William T. Thompkins, Jr., of the MIT Department of Aeronautics and Astronautics. Working with him were staff member Robert Haimes and student Patrick Dirks. This task was designed to produce a methodology for defining special purpose computer architectures for computational fluid dynamics calculations and integrating numerical algorithm development with architecture development. The types of architecture considered were loosely coupled, multiple processors which cooperate on a single CFD task or calculation. The distinction between tightly and loosely coupled architectures refers to synchronization requirements between processors which are largely determined by rules for updating global or shared memory variables. In tightly coupled

environments, a process may only update global memory variables at a fixed point in the calculation sequence; in a loosely coupled environment, processes may update global variables on a non-deterministic basis.

What is commonly meant by integrating algorithm and hardware development is particular coding strategies for particular machines. For example, analysis of implicit, approximated factored codes for the CRAY-1 shows that very little speedup over a scalar processor is available unless one vectorizes across sweep lines for the matrix LU decompositions. Such a result dictates the form that an efficient code must take on that machine using that algorithm. This project paid some attention to the problem of how we should code a given algorithm for a given architecture, but it focused primarily on the question of what algorithm features we should strive for.

During the first part of the grant period, efforts were concentrated on developing a simulation capability for the dedicated minicomputer/array processor/memory system machines being developed for CFD applications (see presentation 1). This machine provides a single user about CDC 7600 processor capacity for mesh sizes of order 250,000 nodes. Appropriate block level units and their timing and combination rules were devised. While the low cost, dedicated machines can always provide an interesting level of performance, these configurations are difficult to push to supercomputer speed levels. The primary disadvantages are demands on the memory management computer and the fact that the I/O bandwidth requirements grow intolerably large as the number of external processors increases. Progress toward general purpose, multiple processor, multiple memory module (MPMMM) configurations has also been quite slow because of processor synchronization and switching system difficulties.

The goal was to develop a simulation capability for multiple processor machines executing concurrent but identical instruction streams and to realistically simulate the performance of particular minicomputer/array processor systems. These goals have been accomplished with the generation of a LISP-based simulation which allows a user to explore the effects of different hardware configurations and operating system policies in Computational Fluid Dynamics (CFD) work. The simulation provides realistic forecasts of actual times required for the execution of a particular series of programs on a given hardware system as well as the relative performance of different systems on a given job.

The simulator was designed first and foremost with generality in mind. As much as possible, the user should be able to simulate a wide range of device characteristics and system policies by making small changes in various parameters, or substituting alternative modules for limited sections of the simulation code. For the former, an understanding of the available "hooks" in the system is necessary; the latter

requires a fuller understanding of the design of the simulation to replace certain pieces without disrupting the overall operation.

The basic model of the processor system is an arbitrary number of Processing Elements (PE's), controlled by a single Central Processor, all connected through a system-wide bus to a number of peripheral devices. Devices currently modeled include Main Memories (MM's), Array Processors (AP's) and Bulk Memories (BM's). Note that any number of controllers (device units) of a particular device type can be connected to the bus. The "System Bus" allows central control of all PE's and all devices by the CPU. In addition, the System Bus is used in all transfers between devices in the system. The PE's are slaved processors; they execute instructions for processes when scheduled by the CPU.

Main Memory represents processor memory available to all processors in the system simultaneously. While Main Memory is limited in size, there is never any contention for access to the device. In addition to Main Memory, there is Bulk Memory, BM. This represents large non-local memory. Like Main Memory, it is limited in size. In addition, Bulk Memories act like ordinary I/O devices: there is a single controller per device unit, and access to a particular Bulk Memory is limited to a single PE at a time. In addition, the transfer speed of Bulk Memory is usually lower than the transfer speed of Main Memory, but there is usually more of it available. Typically, Main Memory is used as a buffer in transfers between Bulk Memories and Array Processor Storage. Finally, there are Array Processors, or AP's. An Array Processor is unique in that it appears as two separate devices: an I/O controller that allows transfers of data into and out of AP memory, and an Array Processing Unit, or APU, which is a local processor that operates on the data in the AP's memory. The AP device allows simultaneous transfers to and from AP memory and computation by the APU on the data elsewhere in AP storage. AP's are typically very limited in storage, but the APU provides very high speed computation compared to the PE's.

By manipulation of a small number of internal parameters, the user can simulate different hardware configurations and hardware characteristics. These parameters allow the user to vary the number of Array Processors in the system, for instance, or the transfer speed of Bulk Memory devices, the total storage available in the Array Processors, or the speed of Processing Elements. Perhaps more interestingly, there are hooks that allow the user to change the behavior of the various devices in the system. Some devices may be modeled as a single controller, and access to the controller limited to a single processor for the duration of a transfer. Others may be modeled as having infinite shared access, so that processes never compete with each other for the availability of a particular access path. Similarly, some devices may be

reserved to a single process, while others may be shared between all processes in the system at all times. The details of the available hooks are described in the chapters on devices and on processors in publication 2.

Finally, the user may also change the behavior of the operating system being simulated itself. There is actually only a very thin line between the implementation of the simulation of the complete system and the simulation of the operating system controlling the system. By changing some functions in the system, the model of the operating system may be changed to simulate different scheduling algorithms, for instance, or different resource allocation strategies.

The simulation of the operating system is an essential part of the simulation as a whole. The entire simulation may be viewed as the simulation of the operation of a given operating system on a particular hardware configuration, faced with the demands of a number of processes in the system. The operating system serves two functions in the simulation: the scheduling of processes on processors, and the allocation of the system's resources to the processes in the system. The goal of the operating system is to maximize the use of the available resources, reducing time spent waiting to a minimum.

The simulator was used to model the performance of the lab minicomputer/array processor system and it produces the correct qualitative system performance. A small effort was made to improve the quantitative system prediction. A Computational Fluid Dynamics Laboratory report was prepared which describes the simulator and its applications to current hardware and algorithms (see publication 2).

Fault-Tolerant Processor Architectures -- Dr. T. Basil Smith of the C. S. Draper Laboratory was initially in charge of this research task. Upon his transfer from the Laboratory, Dr. Jaynarayan Lala took responsibility for the work. Graduate student Richard Harper also participated in this aspect of the program. The objective of this task was to begin development of a fault-tolerant processor architecture that can provide high-throughput general-purpose computation in a software environment which is comparable to that of non-redundant or simplex super-computer environments. The study focused on the incorporation of fault-tolerant processing techniques to mask and correct faults within the context of a state-of-the-art super-computer architecture.

For many real-time control tasks, processing requirements in the millions-of-operations-per-second (current mini-computer) range are satisfactory, or the problem is easily distributed to loosely coupled multi-computer or multi-processor systems. The Draper Fault-Tolerant Processor (FTP), and Fault-Tolerant Multiprocessor (FTMP) are examples of highly reliable machines matched to these needs or

requirements. There also exists a class of problems which requires higher throughput, and in which a loose coupling of processors is inadequate due to the overheads in communicating between the coupled processors. Ground-based super computers such as the CRAY-1 and the CYBER-205 are examples of existing machines which address this need. The dataflow architectures and CFD architectures are alternative architectures, as yet unimplemented, which are also addressed to these needs. All of these architectures employ large numbers of elements or components and exhibit poor reliability and availability in fault intolerant implementations. Availability and maintainability are clearly inadequate for many real-time scenarios or spaceborne applications. MTBF is typically on the order of tens of hours.

The architecture of existing "super-computers" such as CRAY-1 and CDC STAR series computers were examined from the viewpoint of incorporating "synchronous fault tolerances" into these machines. The highly pipelined architecture of such computers has been the only commercially successful approach to achieving very high throughput. This type of architecture may be combined with the concepts underlying the fault-tolerant computers developed at Draper Lab for avionics applications. The basic concept here is one of synchronous operation of redundant hardware elements such that their results can be matched on a bit-for-bit basis under no-fault conditions.

It was then decided to focus on the design of a computer architecture which might be suited for the rapid solution of problems which involve the evaluation of a large set of alternatives and the estimation of the consequences of many possible courses of action. Examples of problems of this sort lie in the areas of decision support systems, autonomous systems control, route planning, and other fields. We call such problems "generalized planning problems."

One view of such problems is to consider the set of decisions which may be evaluated by the planning program to comprise a tree structure which must be searched for a path which results in the optimization of a cost criterion. For example, the Dijkstra algorithm for finding the minimum cost path through a graph is equivalent to a degenerate form of the A* tree searching algorithm. As another example, the traveling salesman problem (TSP) has been cast as the search of a tree structure, where each node of the tree represents a set of possible solutions of the problem. The search procedure corresponds to the continual refinement of these sets until a satisfactory solution is found. As a final example, another solution method for the TSP is the route construction method. Here, each node of the tree corresponds to a partial solution to the problem.

Preliminary considerations indicate that problems which can be cast into such a tree search formalism can potentially possess a high degree of parallelism, on both the coarse and

fine scale. This parallelism can be efficiently exploited by a multicomputer system, and thus solutions to such generalized planning problems can be obtained much more quickly than when they are attacked on a sequential computer.

A preliminary proposal for such a multicomputer system has been generated. The system consists of a loosely coupled set of processing elements (PE's) which communicate with each other via an intercomputer network. The use of many computers to solve the problem exploits the large-scale concurrency which resides in the tree search formulation of the generalized planning problem. Each PE is envisioned as a heavily pipelined Multiple Instruction Multiple Data (MIMD) machine, which can further take advantage of the additional fine-grained parallelism inherent in the problem. The PE will have a reduced instruction set to simplify its design, programming, and operation, and to help speed up its instruction execution rate. A report describing this proposed architecture has been prepared (see publication 3).

For this computer to be adequately reliable for mission and life-critical missions, it must satisfy theoretically demonstrable architectural requirements. Specifically, it must be capable of tolerating arbitrary failure behavior of a subset of its components with near-unity coverage. The only known way to achieve this is via synchronous redundant bit-for-bit replicated computation, which in turn hinges upon the capability to provide non-faulty redundant sites with bit-for-bit inputs in the presence of such arbitrary failure modes. This problem is the well-known Byzantine Generals problem. For a solution to this problem to be possible, four basic architectural requirements must be met. First, the redundant sites must be partitionable into 3f+1 fault containment regions, where f is the number of simultaneous faults it is desired to tolerate. Second, the fault containment regions must be interconnected via 2f+1 disjoint paths. Third, the fault containment regions must perform a minimum of f+1 "rounds" of communication, where a round may be loosely defined as an algorithmic phase in which each fault containment region communicates with each other. Finally, the execution rates of the different fault containment regions must differ by at most a known upper bound, i.e, the fault containment regions must be synchronized. These are absolute minimal requirements which must be met by any system hoping to be highly reliable, and we have concluded that the architectures that we have surveyed do not explicitly meet these constraints. Therefore, we must study how to incorporate these requirements into an architecture which meets the functional requirements of the application.

Dataflow Techniques -- In charge of this effort was Professor Arvind of the MIT Department of Electrical

Engineering and Computer Science. Working with him was graduate student Research Assistant Gregory Papadopoulos. This task investigated a dataflow approach to highly reliable computation. Drawing from dataflow principles, it is suggested that an automatic procedure may be developed which can transform a failure-sensitive graph of unreliable operators into one which can tolerate random operator failures yet completely implement the algorithm specified by the original, unfailed graph. It is expected that at least two advantages will be gained by this approach. First, synchronization requirements inherent in present fault-tolerant systems will be relieved, allowing easier expansion and distribution with the commensurate reduction of the risk of synchronization failures. Second, the isomorphism of dataflow languages with computation graphs promises a much more rigorous basis for the wanting area of software validation for highly reliable systems. On balance, dataflow seems extremely well-suited for the formulation of fault-tolerant systems.

We are able to apply sets of transformations to any data flow program written to execute on a perfect machine and create a model of a system which can tolerate a specified number of hardware failures. The model contains the familiar concepts of voters and redundant program copies along with abstractions of the important features of the underlying hardware: the number of independent modules, and the nature and topology of their interconnection. Transformations have been developed to (1) provide redundant modules with identical or congruent copies of all input data, (2) correctly support time-out tests and associated nondeterministic dataflow operators, (3) interactively bound the time skew of similar computations at redundant sites. These transformations come in two forms: ones that minimize the total number of independent modules required, and ones that minimize the total amount of information exchange between modules.

In addition to unifying the hardware and software under a single analytic paradigm, the transformations inherently lack any requirements for synchronous and/or centralized control -- making the models well-suited for very large, distributed computer systems.

Our goal was to take an arbitrary (but well-formed) dataflow program, the simplex source, make it redundant, and impose the necessary restrictions on the physical implementation so that it is fault tolerant. The source program is given in the language L_0, a deterministic language with fix-point semantics, and the programmer assumes that it will operate on a perfect machine. Programs are automatically transformed into redundant, fault-tolerant ones as described by the language R_0. By abstracting elements of an implementation, *fault sets*, which correspond to processing sites that fail independently of each other, and interconnection links, which correspond to data paths between these processing sites, we are

able to capture the nature of modular redundancy. Our assumed fault model, token fragments, is an extremely general one and basically states that a faulty element can transmit any data (or no data at all) in any fashion to all elements which are connected to it. Additionally, two receivers of what would otherwise be identical data (from the same bus wire for instance) could interpret different values for data transmitted by a faulty source. As discovered by other researchers, primarily the NASA-sponsored work of the SIFT project at SRI and the FTMP project at C. S. Draper Laboratories, an essential problem in the correct implementation of modular redundancy is the ability to consistently or congruently distribute copies of a simplex sensor to all redundant processing sites. Data flow readily exposes this problem as well as providing concise, graphical solutions in the form of congruence schemata. For the multiple simultaneous fault case, we give recursive formulas for schemata which require a minimum number of fault-sets for their implementation as well as ones that require a minimum amount of information exchange.

We then extend our source language to one that expresses non-determinism. This gives us the ability to perform time-out tests of simplex data sources, support non-deterministic merges and stream-based programming, as well as an understanding of synchronization, scheduling, and clocks. We define an event as the receipt of a token (packet of data) by an operator. To correctly support non-determinism we argue that a consistent time total ordering of selected events must be derivable by all redundant processing sites. Schemata, very similar to congruence schemata, which can support such an algorithm are derived. Finally, we show how to bound the time skew between similar events occuring at redundant sites. This gives a notion of clocks and synchronization which as readily describes the microsynchronous clocking of the FTMP as well as the frame synchronization strategy of SIFT, but in what we believe to be a more general and perspicuous fashion.

A discussion of all these issues has been prepared in the form of a paper that was presented at the 1984 American Control Conference (see publication 4).

Software Specification and Verification Tools -- The work on this research task was performed by Dr. Frederick Furtek of the C. S. Draper Laboratory. For both autonomous and manned spacecraft, reliable software is crucial to successful mission performance. The complex and intricate nature of the real-time software found on most spacecraft, however, makes the task of achieving reliable software extremely difficult.

The reliability of software can be enhanced through techniques applied either (1) during design/coding, (2) during testing, or (3) during online operation. Techniques applied during the first two phases increase reliability through fault

avoidance, while techniques applied during the third phase increase reliability through fault detection and recovery (fault tolerance). This task concentrated on achieving reliable software through both fault avoidance and fault tolerance.

To avoid software faults in operational code, it is crucial that the reasoning underlying program operation be formalized and verified. This effort is essential since all software is developed through the reasoning of programmers, and it is this reasoning, in lieu of exhaustive testing, that is the ultimate source of confidence in the correctness of software. This task sought to develop a capability in software verification that does not presently exist but which may be achievable by drawing upon and extending present verification techniques.

An in-depth survey was performed of a number of existing techniques for software specification, with emphasis on approaches suited to describing real-time, concurrent behavior. Advantages and shortcomings of each approach were noted, and, as expected, there was no single technique that met all of our needs. In particular, there was no approach that could adequately represent concurrency, indeterminacy, and timing dependencies and that could provide specifications at different levels of abstraction. All these are essential requirements for specifying complex, real-time, distributed systems.

The abstraction requirement proved to be especially troublesome in formulating a new approach since there are several dimensions along which a system can be decomposed. Successive levels of details can be added in three interdependent ways:

(1) By elaborating on the definition of functions. That is, by defining complex functions in terms of ever simpler functions.

(2) By elaborating on the definition of data structures. That is, by defining data structures in terms of ever more primitive data structures.

(3) By describing behavior at finer granularities of time.

While there exist well-understood techniques -- applicative languages -- for dealing with the first two types of abstraction, very little has been done with time abstraction.

Since all three types of abstraction are essential in describing complex real-time systems, our efforts focused on integrating the three abstraction techniques into a coherent framework. More specifically, we attempted to incorporate the three techniques into a previously developed specification language.

The "Specification and Verification" task achieved one of its principal goals: a framework for rigorously specifying the behavior of real-time, distributed systems. The framework is based on the widespread view of a concurrent/distributed system as a collection of interacting processes. Unlike previous approaches, however, no assumptions are made about the mechanisms for process synchronization and communication. One is able to describe the behavioral constraints imposed by such mechanisms without being forced to consider the details of process interaction. A key element of the framework is a formal language that permits the expression of a broad range of logical and timing dependencies, many of which are inexpressible with existing techniques.

The technique incorporates a number of conceptual and technical advances:

- A formal systems model has been developed that accommodates multiple processes interacting both synchronously and asynchronously. This capability is made possible by a novel approach that separates a specification into a synchronic part and a logical part.

- A new language for describing complex logical and timing relationships has been developed. The language, whose formulas resemble Boolean expressions, is based on the five logical primitives: 'not', 'and', 'and_next', 'and_next*' and 'reverse'. Through the use of higher-level constructs, it is possible to write specifications in a form that approaches natural English.

- A way has been found, using the syntax of the Ada programming language, to integrate the concept of abstract data type into the specification technique. The specification approach can thus be seen as a way of making formal, implementation-independent statements about the intended behavior of Ada objects.

These results are described in publication 5.

Work remaining on the development of the specification framework falls into three main areas:

- Improvements and Extensions -- The need to improve and extend the framework will inevitably arise as applications experience is gained. One area in need of improvement that has already been identified is the synchronic structure, which is presently limited in the sorts of synchronic relationships it can express.

- Verification Capabilities -- Although the desire to rigorously verify system behavior has provided much of the impetus for the present effort, the issue of verification has not been addressed. Progress in this area has been made under prior funding, but substantial work remains.

- Hierarchical Specification -- Composing (or decomposing) a specification in a hierarchical fashion is the most effective way of dealing with complexity. The appropriate mechanisms for 'connecting' different levels of a heirarchical specification need to be explored.

Management of Software Development -- Professor Stuart Madnick of the MIT Sloan School of Management was responsible for this research task. Graduate student Research Assistant Tarek Abdel-Hamid also participated in this effort. The past decade has witnessed the development of a large number of software engineering tools and techniques for improving the development of software systems. During the same time, research in the behavioral sciences continued to produce dozens of other tools and methods to improve organizational functioning and effectiveness. As a result of these developments, software project managers today have at their disposal an abundance of sophisticated tools that are potentially useful in helping them maintain or increase organizational effectiveness.

The objective of this research effort was not to provide another specific software engineering tool. Instead, it offers an integrative perspective on software project dynamics that can help managers answer the difficult questions they need to raise when assessing their organizations' health, selecting improvement tools (from the many that are already available), and implementing their choices.

To accomplish this, we developed an integrative system dynamics model of software project management dynamics.

However, using an integrative model merely to "alert" managers to all the important aspects of a problem, while clearly useful and essential, is definitely not enough. Because such a model will undoubtedly contain a large number of components with a complex network of interrelationships, we must, in addition, provide an effective means to determine both accurately and efficiently the dynamic behavior implied by such component interactions. And this, it turns out, is quite challenging to achieve.

Experience from working with managers in many environments indicates that they are generally able to specify the detailed relationships and interactions among managerial policies,

resources, and performance. However, managers are usually unable to determine accurately the dynamic behavior implied by these relationships. Human intuition, studies have shown, is ill-suited for calculating the consequences of a large number of interactions over time.

By utilizing the simulation techniques of system dynamics in this research effort we, thus, combined the strengths of managers with the strengths of the computer. The manager aids by specifying relationships within the software project management system, the computer then calculates the dynamic consequences of these relationships.

Considerable progress was made toward the objective stated above:

First, we completed the development of an integrative system dynamics model of software development project management. The model complements and builds upon current research efforts, which tend to focus on the micro components (e.g., scheduling, programming, productivity,... etc.), by integrating our knowledge of these micro components into an integrated continuous view of the software development process.

Second, a case-study was conducted at NASA's Goddard Space Flight Center to test the model. The model was highly accurate in replicating the actual development history of the DEA software project. Project variables tracked included: the workforce level, the schedule, the cost, error generation and detection, and productivity.

Third, the model was used as an experimentation vehicle to study/predict the dynamic implications of an array of managerial policies and procedures. Four areas were studied: (1) scheduling; (2) control; (3) quality assurance; and (4) staffing. The exercise produced three kinds of results: (1) uncovered dysfunctional consequences of some currently adopted policies (e.g., in the scheduling area); (2) provided guidelines for managerial policy (e.g., on the allocation of quality assurance effort); and (3) provided new insights into software project phenomena (e.g., Brooks Law). These matters are discussed fully in publications 6 and 7.

The Software Environment for Concurrent Computing -- This work was led by Virginia Klema of the MIT Concurrent Computing Group. Working with her were staff member Elizabeth Ducot and graduate student Richard Kefs. This work has been augmented by the close cooperation of Professor George Cybenko of the Tufts University Department of Computer Science and his graduate students David Krumme and K.N. Venkataraman.

One of the stated points of focus of the NASA initiative in computer science is concurrent computing. The Concurrent Computing Group at MIT has for some time been studying the issues involved in use of concurrent computing capabilities to execute demanding algorithms. This work includes study of the

details of hardware organization, the design of algorithms intended for execution on a collection of computers, and the environment in which the necessary software is developed. The computing environment which is used for experimental support of this work is pictured in the attached figure.

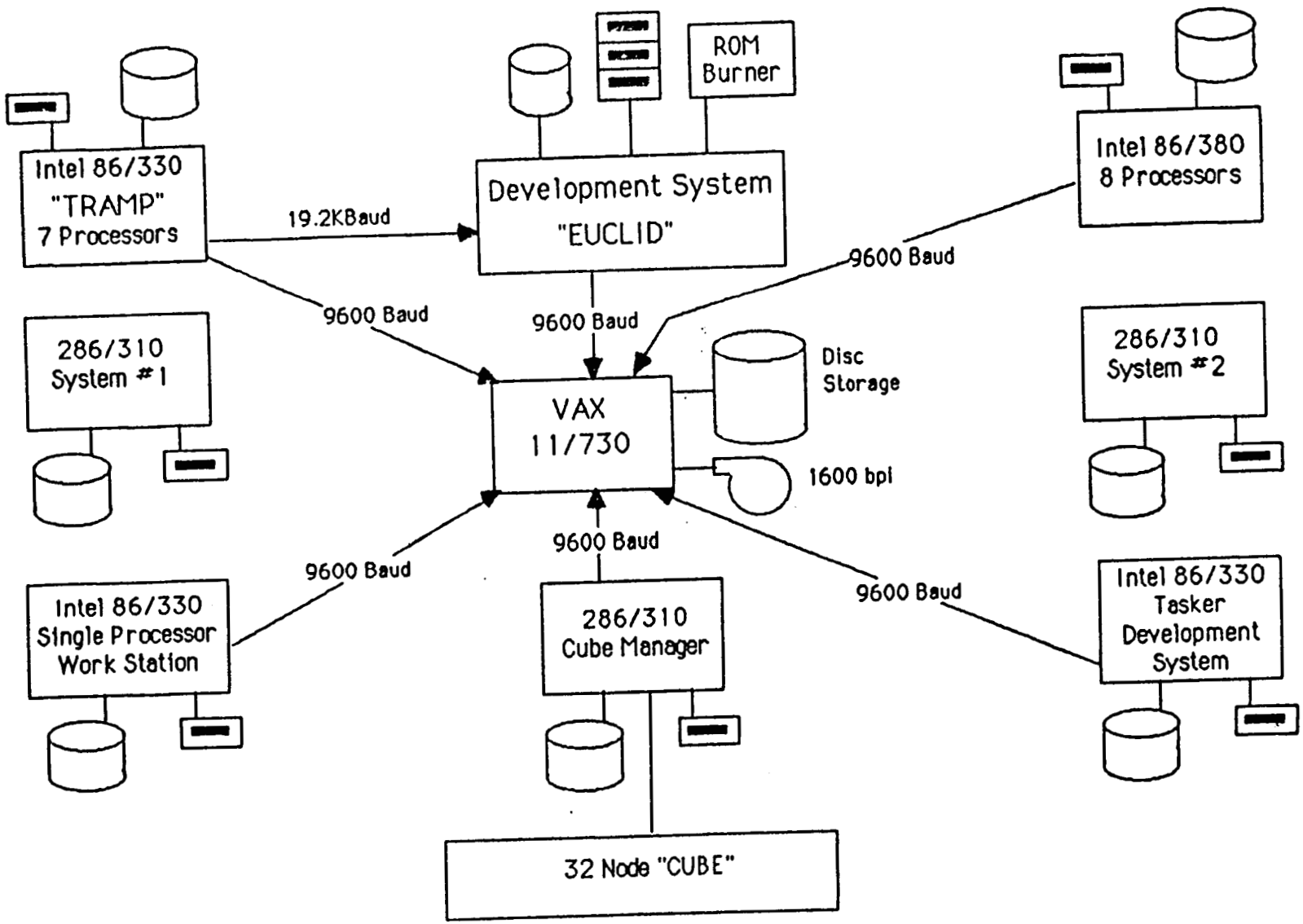
The work under this grant was concentrated on the software environment for concurrent computing with microprocessor-based systems. A major accomplishment has been the first-draft definition of a concurrent tasker to interface with the Intel iRMX operating system. This provides the vehicle that installs code and data segments in the distributed computers and monitors execution on and among the computer processing elements. The intent of this tasker is to provide the user a convenient environment in which to specify his concurrent computing process. A summary of this initial effort is given in publication 8.

Other research related to multiprocessor systems and parallel computational algorithms is reported in publications 9 and 10.

Parallel Algorithms and Architectures for the Solution of Partial Differential Equations -- Professor Bernard Levy of the MIT Department of Electrical Engineering and Computer Science was responsible for this research effort. Working with him were Professor Bruce Musicus and graduate student Jay Kuo, also of this department.

The main focus of this research was to develop parallel algorithms and architectures for the numerical solution of partial differential equations. (PDEs). The effort was focused primarily on elliptic PDEs and on the use of mesh-connected arrays of processors, or pyramidal arrays, for their solution. Most schemes which have been proposed up to this point are only parallel implementations of PDE algorithms which were developed for Von-Neumann (single processor) computers. In contrast, we have tried to develop algorithms which take full advantage of the parallelism which is provide by multiprocessor architectures. The main feature of these algorithms is that their communication requirements have to be taken explicitly into account when evaluating their complexity. To see this, note that global communications on a square array of NxN processors take O(N) time, whereas local communications between a processor and its nearest neighbors take only one time unit. This has motivated the development of a local relaxation method of solving elliptic PDEs, where instead of using a single relaxation parameter as in David Young's successive overrelaxation (SOR) method, we use a set of local relaxation factors which are determined locally. This method was shown to converge, and even on a single processor computer, it is faster than SOR. Furthermore, it can be implemented in parallel, whereas the SOR method is not parallelizable, so that

CONCURRENT COMPUTING LABORATORY



on an array of M processors, it is at least M times faster than SOR. This method is described in publication 11.

Work has also progressed on the development of parallel multigrid algorithms for the solution of elliptic PDEs. Multigrid algorithms are the fastest of the algorithms currently available for solving elliptic PDEs on single-processor computers. Furthermore, they are based on a relaxation principle, so that they are parallelizable. The only difficulty which has not yet been overcome is to figure out how to run all grids (coarse and fine) in parallel. The parallel multigrid technique that has been developed relies on the observation that each grid can be viewed as computing a different frequency band of the solution, and that the transfer from a fine grid to a coarse grid can be viewed as a band pass filtering operation. From this point of view multigrid methods are similar to multirate digital signal processing techniques, and we have been using this insight to develop the parallel multigrid procedure. In addition, we have developed a frequency dependent finite-difference discretization technique which can be used to discretize a given PDE optimally at each grid level. This method is currently being implemented, and the results will be described in a technical report.

Publications and Presentations

1. Thompkins, W.T. and R. Haimes: "A Minicomputer/Array Processor/Memory System for Large-Scale Fluid Dynamic Calculations," Presented at the Symposium on the Impact of New Computing Systems on Computational Mechanics, ASME Winter Annual Meeting, Nov. 1983, Boston, MA.

2. Thompkins, W.T. and P.W. Dirks: "The Nemesis System: Simultaneous Simulation of Computer Architecture, Numerical Algorithms, and Operating Systems," MIT Computational Fluid Dynamics Laboratory Report CFDL-TR-84-5, September 1984.

3. R. Harper: "Computer Architectures for Decision Support Systems," C. S. Draper Laboratory report, September 1984.

4. Papadopoulos, G.M. and Arvind: "Dataflow Models for Fault-Tolerant Control Systems," Proceedings of the 1984 American Control Conference, June 6-8, 1984.

5. Furtek, F.C.: "Specifying the Behavior of Concurrent Systems," C. S. Draper Laboratory Report CSDL-P-1915 July 1984.

6. Abdel-Hamid, T.: "The Dynamics of Software Development Project Management: An Integrative System Dynamics Perspective," PhD thesis, MIT Sloan School of Management, December 1983.

7. Abdel-Hamid, T.K. and S.E. Madnick: "The Dynamics of Software Project Scheduling: A System Dynamics Perspective," Communications of the ACM, May 1983.

8. Ducot, E.R.: "Application Interface to the Concurrent Environment," MIT Concurrent Computing Laboratory Report, July 1985

9. Krumme, D.W. and R. Kefs: "Implications of Shared Memory for Real-Time Operating Systems," Informal abstract of research done in cooperation between the Department of Computer Science, Tufts University, and the Concurrent Computing Group, MIT., 1985.

10. Krumme, D.W., K.N. Venkataraman, and G. Cybenko: "Hypercube Embedding is NP-Complete," Technical Report 85-1, Department of Computer Science, Tufts University, August 1985.

11. Kuo, C-C, B.C. Levy, and B.R. Musicus: "A Local Relaxation Method for Solving Elliptic PDEs on Mesh-Connected Arrays," MIT Laboratory for Information and Decision Systems Report P-1508, October 1985. To appear in the SIAM Journal on Scientific and Statistical Computing.